VZ-LINK NEWLETER

<< AUCKLAND VZ 300/200 USERS CLUB >>

ISSUE # 19 APRIL 1988

I am pleased to report that the meeting/workshop,
held on the 27Th Feb...went really well.
We had 20 people attend and this made it totally worth while.

My Thanks to the following people:

Rev Bob Ferguson for the use of the hall.
Brooke Spong and Danny Todd for bringing their VZ equipment.
Graeme Fisher for showing the younger members some programming skills,including Chip 8.
Peter Patterson,for again lending me his Laserlink software to demonstrate.

And Thanks also to the other ladies and gentlemen who came along to show their support for the club.

We had 3 VZ computers setup and running,
Members were able to demonstrate some of the software they had written or brought.
There was also a fair amount of software exchanged and it was good to see names and addresses swapped.

The next MEETING/WORKSHOP will be held on:

28Th May 1988
Starting at 10.30
Owairaka Baptist Church
Richardson Road,Mt Albert.

This issue includes the Second and Final part of Bob Kitch's 'Introduction to Programming' Item.
We also have a Program called "World Times" written by Richard Davis.
It is rather long,but it is worth taking the time to enter.
If you add any more city's send the additions to me and I will put them in the newsletter.

We also have a item from Joe Leon which shows how to install a 'Shift Lock Switch'.

On page 15 we have a High Scores list from Scott le Brun's " Vz Down Under "Newsletter.
If you have any higher scores send them to me and I will forward them to Scott.
I will also accept scores for our own High Scores list,which will start next issue(space permitting).

So until next issue,happy computing the VZ way.

INTRODUCTION TO PROGRAMMING - Part 2 - by Bob Kitch.

As mentioned in Part 1 of this series, the programming task is a large and complex feat of organization and requires a wide range of skills. It is possible, and best, to break the task down into six segments - each of which must be thought about, planned and then carried out to ensure the successful completion of a software project. Even a small program requires that a cursory consideration of the six segments be made - although some of them may be quickly passed over as trivial. But it is certain that larger programs (more than 200 lines) require careful planning for success.

Before describing the six steps, it is worth thinking about "What makes a GOOD program?"

A program may be judged from a number of different standpoints; each is not necessarily mutually exclusive and sometimes some conflicts require that a trade-off be made.

The first criteria is that a program should be EFFICIENT. Efficiency can be considered from a number of varying view points. For example, optimization of the run-time can be considered as efficient. Also, reduction in storage requirements for both program code and variables can be considered as efficient programming. Furthermore, and particularly if one is developing software commercially, then efficiency can be measured in terms of the actual time required to get an applications program running and the ease of maintenance of that code. The use of appropriate data types and data structures can greatly improve the efficiency of a program. The selection of a suitable algorithm can also assist. Finally, ease of debugging so that the program can be updated or modified may be considered desirable.

The second criteria is GENERALITY and it is here perhaps that so many programs "score" so poorly. Rather than a program being written to solve a particular chore, it should be broadly written to handle a wide range of problems. The use of subroutines and functions developed and debugged previously can enormously, improve programming productivity. Often a simple substitution of a variable for a constant in a program can broaden the the applicability of the program significantly.

The final criteria is ELEGANCE, which is a little harder to both define and achieve. An elegant program is one that is simple and ingenious, and possibly uses an algorithm or data structure that may not be immediately obvious to the application. The so-called "programmer's tricks" are often elegant solutions to a programming problem; but beware, some are attempts by programmers to conceal their programming stategy.

These then, are general guidelines to try and attain in your programming and by which to judge a particular programming effort as good, mediocre or poor. Notice that they are not language specific comments and are equally applicable to any programming language or exercise.

To return to the six steps in the programming task - I will briefly discuss each in turn and ask that you consider each one when embarking upon your next programming exercise. Also as one proceeds through the steps, it is often necessary to recycle back through some of the preceeding steps, to iteratively improve the exercise and your understanding of ideas.

1. PROJECT SELECTION. This may appear trivial, but we all have too many ideas for programs and rarely know which one to tackle next. Also be honest with yourself; some of the projects are probably too ambitious for your existing skills and an attempt upon these will possibly result in frustration and perhaps failure. Choose an exercise that is challenging and worthwhile. Try not to "reinvent the wheel", try to be aware through reading magazines or discussing with other Users what programs are already available. Modifying an existing program to suit your specifications is sometimes quicker - it also allows you to study how other programmers tackle problems. O.K., so now you have an idea or problem that you wish to tackle and solve.

2. PROJECT FEASIBILITY. Again be honest. Do you have the hardware, software and know-how to achieve the result? Its is not really much use trying to write large business-oriented data base programs for an 8K tape-based VZ! Check that the task is reasonable.

3. PROJECT DEFINITION. This is where the idea starts to get translated into a reality. It is also the phase where generality can be written in. It is easiest to start by thinking about the input to the program. Is it keyboard oriented, or is it to come from a programmble I/O port? Perhaps the program reads only DATA statments to configure itself or maybe the program must check if a printer is connected to the sytem? Start defining what the input will look like. Assign variable names with meaningful mnemonic names at this stage also.

Next, define the output expected from the program. Is it to write to tape and in what format? Perhaps it is to be screen oriented - can sound be used - or perhaps voice synthesis to tell the operator what is going on? Plan very carefully and fully the layout of the expected output as this is how Users will initially perceive the quality of the program.

After defining the I/O for the program we should now have a feel for the anticipated range of parameters that the program is meant to accept and also handle. This brings in the very important concept of defining the BOUNDS within which the program must function correctly. Following on from this, is range checking of all input parameters so that the program cannot go beyond the range that it was designed for and give unexpected results. A number of warning messages must be built into the program along with error capture and recovery routines. It is failure to define the operating bounds of a program that causes most crashes or rogue behaviour. Even the definition of integer variables at this stage can assist by improving program execution time and reducing storage requirements.

The definition stage should be roughed out on pieces of paper kept for later reference. Perhaps better, is to use an old exercise book. Another benefit of this is that over a period of months your progress can be measured and your growth of programming ideas recorded. Another benefit (although I hardly dare mention it!) is that if, after the coding stage, a system crash occurs and you didn't SAVE the program, then all is not lost - at least an outline of the program remains.

4. DESIGN PHASE. Having sorted out I/O and operating bounds, the actual selection of an algorithm to achieve the result is commenced. By this time some idea of the number of variables required and their type should have begun to gel. This is also the stage where your basic honesty in stages 1 and 2 may catch up with you! Data structure organisation and algorithm selection are really experience-related skills.

INTRODUCTION TO PROGRAMMING CONT. :-

Hence the suggestion to read and/or modify existing programs. But do not despair - practice makes perfect.

5. IMPLEMENTATION PHASE. To date very little actual coding should have been done; in fact the computer need not even have been turned on! Some people may be surprised at how late in the task the computer actually enters into the picture. An awful lot of planning and organizing can be done off the computer and on the "backs of old envelopes".

It is also at this stage that the choice of programming language should be made. Is the program time dependant? If it is, then it should probably be written is Assembler. If the actual timing is not so critical then writing in BASIC with its diagnostics and helpful features (so typical of a high level language) deem it sensible. Experienced programmers will probably use a bit of each in practice. A very sensible compromise is to develop the program in interpreted BASIC and once finalized and debugged, compile the BASIC code to speed up execution.

6. EVALUATION PHASE. This is the moment of truth! Does the program fulfill all the criteria set out in the definition phase. If so, then you have succesfully achieved your task. Is the output as you expected it? Are the results correct? It is a good idea to have a standard set of data to exercise the program so that it can be quickly verified after a program alteration. Ensure that all logical paths through the program have been exercised so that no spurious errors of logic remain undetected. Finally, deliberately try values that are out of the intended bounds of the program to ensure that you have trapped them and that the program recovers from this type of misuse above and beyond its' intended design range.

As mentioned in part 1, the offer on programming queries still stands. If writing to Bob Kitch please include a SASE.

BOB KITCH (07) 378 3745
7 EURELLA STREET
KENMORE 4069
QUEENSLAND AUSTRALIA

VZ ASSEMBLY

Although you may not realise it,BASIC has some very handy instructions.One begins to take its high-level commands for granted,and when starting Assembly language,one is often stuck with a problem which seems unsolvable !.
Imagine,for example,how often you use the FOR-NEXT combination in BASIC.However,when it comes to Assembly language programming,such a loop is not so easy!.

Despite the fact that the Z80 does not offer us a FOR-NEXT statement,it does provide a very similar instruction-DJNZ.
DJNZ stands for 'Decrement & Jump if not zero'.To understand exactly how it works,let's look at an example.Suppose we wish to sound the 'beep' in the VZ 20 times:

```
LD B,20
PUSH BC
```

CONTINUED ON NEXT PAGE

```
LOOP CALL 3450H        ;Sound 'beep'
     POP BC
     DJNZ LOOP         ;& Loop 20 Times
```

Firstly,the B register is loaded with the number of loops. The beep is sounded by CALLing the ROM routine.This is labelled 'LOOP'.
The DJNZ instruction follows;notice it has the form:DJNZ (label). At this point,the CPU decrements the B register,and examines the result.If it is not zero,the CPU jumps back to 'LOOP' and the beep sounds again.After the B register reaches zero,the CPU simply goes on to the next instruction,Note that as the B register can only be used with this instruction,a maximum of 255 loops is available.

Obviously,there will be times when you will require a much larger number of loops.This can be done,still using the DJNZ instruction,via nested loops.Heres an example:

```
      LD B,255         ;set high byte
LPA   PUSH BC          ;& save on stack
      LD B,255         ;set low byte

LPB   (insert main program here)

      DJNZ LPB
      POP BC
      DJNZ LPA
```

So,we see that the DJNZ instruction can be used rather like the FOR-NEXT of basic.

You'll recall that last issue we looked at Status Flags,and specifically at the Zero flag.Let's look at a few other flags and when and where to use them.

The 'N' flag can be a little confusing to programmers who have used micro's other than the Z80,since it usually means 'Negative'. However,in the Z80,it is used for some BCD (Binary Coded Decimal)operations.Its status cannot be tested directly by the programmer,and so it is of academic importance only.

Before we can look at one of the most important flags,the carry flag,we need to go back to the basics of the binary number system.Suppose we want to add the number 128 to itself,in binary (which is of course,how the Z80 must do it).We'll carry out the sum by column addition:

```
          10000000 +
          10000000
          --------
Answer=  100000000 (256)
```

As we move from right to left,the columns simply add to give zero,except at the eighth bit,where two 1s give 2 (10 in binary). Just as would be expected when working with decimal,the '0' is put down under the column,while the '1' is carried to the next column.But you say "This is eight-bit binary...there is no ninth bit!".
To prevent this 'ninth bit' being lost(which would make the answer 0!),it is placed in a special flag called...you guessed it!...the carry flag.Now let's look at a practical example of how it is used:

```
        LD B,0          ;set MSB to zero
        LD A,(NUMA)     ;get first number of add in A
        LD D,A          ;now in D
        LD A,(NUMB)     ;second number in A
        ADD A,D         ;add them
        LD C,A          ;answer in C
        JR NC,EXIT      ;jump if no carry
        INC B           ;else inc hi-byte
EXIT    RET             ;and return
```

This is an example of a routine to add two eight bit numbers together. The numbers are stored in the respective memory locations NUMA & NUMB. The result of the addition is placed in the BC register pair. The first number is placed in the D register, while the second is left in the A register. The ADD instruction is used to add the two registers, placing the result in A. Apart from simply adding the two eight-bit numbers, the ADD instruction also sets the Carry flag according to the 'ninth bit'. If the result is less than 256, then it will not have a 'ninth bit', there will be no carry, and the routine will return with the answer in the C register (BC really, as B will still contain zero). However, if there is a Carry, then this means the answer is 256 plus the contents of C. Thus the B register (the high-byte) is incremented, and the answer will be in BC.

This is just one example of how the Carry Flag is used. By the way, the user has direct control over the Carry flag, being able to set it with the SCF (Set Carry Flag) instruction, or complement it with the CCF instruction. (To zero the flag from an unknown condition, do a SCF then a CCF).

The H, or Half Carry flag works in a similar way to the Carry Flag, except that it is used to signal a carry from the third to fourth bit. This is an important feature when using BCD, and we'll be looking at this whole area later.

That's about it for this month. Next month we go on to look at addition, subtraction, multiplication and division with larger numbers.

POKING AROUND WITH THE VZ

```
TO CHANGE TO A INVERSE SCREEN-----POKE30744,1
TO CHANGE TO NORMAL SCREEN--------POKE30744,0
TO STOP KEYBOARD BEEP-------------POKE30779,0
TO RESTORE  K/B  BEEP-------------POKE30779,1

TO FIND TOP OF MEMORY-------------PRINTPEEK(30897)+256*PEEK(30898)

TRACE ON--------------------------POKE31003,173
TRACE OFF-------------------------POKE31003,0

TO PRINT THE DISK DIR-------------POKE30876,1:DIR
  (THIS WILL OUTPUT TO PRINTER)

TO REBOOT DOS---POKE30862,0:POKE30863,145:PRINTUSR(0)

I HOPE TO HAVE A ITEM ON EXTENDED BASIC NEXT MONTH
```

VZ-EPSON PRINTER PATCH V1.4 —— AVAILBLE FROM VSOFTWAREZ

The VZ-EPSON PRINTER PATCH allows EPSON or EPSON compatible printers to LPRINT or LLIST all of the VZ's INVERSE and GRAPHIC characters. The LO-RES and HI-RES screens can also be dumped to the printer as well.

This latest Version 1.4 has a few extra enhancements over previous versions which I found very useful. This Journal would look dull if I didn't make use of the facilities offered by the Patch. All the program LISTings, screen dumps of both HI and LO-RES screens were done by this Patch.

Before the Patch like many VZ users I used to go right through most programs and remove all INVERSE and GRAPHIC characters so my printer would print out the programs. The only printer on the market that used to be able to print out the VZ character set was the GP 100 which is no longer availble.

INVERSE and GRAPHIC CHARACTERS :-

The Patch handles them with ease with improved character form.

LO AND HI-RES SCREEN DUMPS :-

As there are many different printers on the market which sometimes don't print out the same and have different LINE FEED characteristics. For that reason the COPY command was given extra parameters, COPYn and COPYAn, where n can be from 4 to 8.

The COPYA command is very usefull for LO-RES screen dumps and is ideal for printing program menus. When typing in a menu from a program sometimes it's a bit hard to know what it should look like. The Mental Maths and Mailing List Update screen dumps of their respective menus show what the menus should look like. With my AMSTRAD DMP-2000 printer I used COPYA8 for LO-RES menu screens.

NOTE - A one dot wide border is printed to denote borders.

COPYA5 can be used for HI-RES screens.
(See the screen dump I did for twister in the Feb issue for a example of this)

LTAB(n) - PAGE(n) - FEED(n) :-

These 3 commands are very useful with program LISTings.

LTAB(n) - n = 0 to 63 - This command is used for setting of left margin and works with LLIST, LPRINT and COPY commands. It can also work with DIR and STATUS as well by directing the VZ to print to the printer instead of the screen. EG. :-

LTAB(10):POKE 30876,1:DIR:STATUS

If you want to use another TAB position then just type in your new TAB setting. There is no need to zero your previous TAB setting as the LTAB(n) command does it for you automatically. LTAB(45) is the same as LTAB(0):LTAB(45).

PAGE(n) - n = 0 to 127 - FEED(n) - n = 0 to 127 :-

These two commands set the PAGE length (Number of lines per page) and perf skip. FEED(n) will be ignored unless it is less than page length. The 3 commands used together are ideal to LLIST your programs. EG. :-

```
LTAB(10):PAGE(66):FEED(6):LLIST
```

Using my printer and computer paper the above LLISTS my program to the printer with a left margin of 10, 66 lines per page and perf. skip of 6 lines. This assures me that no text is printed on or near the perforations. The setting for your printer and paper may be different than mine. Test out all the functions of the patch so you understand and are familiar with it's operation.

It's good practice to ZERO the PAGE and FEED commands after use as it's easy to forget or you could end up with a form feed in the middle of the page. Setting FEED(0) to zero turns of auto form feed.

LTAB(n), PAGE(n), FEED(n), LLIST, LPRINT and both COPY commands can be used in direct mode or from within your program, but will only work properly while the patch is present. The Patch will also recognise all the hidden commands which may have been entered using an extended basic utility. Steve Olney's EXTENDED BASIC can also be used with the patch, but is not dependant on it.

VERDICT :- Top marks to Larry Taylor for an exellent utility. I could'nt imagine being without it. Ed.

```
MENU MENU
-------------------
1) - RANDOM ADDITION
2) - RANDOM SUBTRACTION
3) - RANDOM TIMES TABLES
4) - NORMAL TIMES TABLES
5) - QUIT MENTAL MATHS
```

```
RANDOM TIMES TABLES
TYPE 999 TO EXIT
5 TIMES 12 = ? 60
2 TIMES  7 = ? 14
9 TIMES  3 = ? 28
TRY AGAIN WRONG ANSWER
9 TIMES  3 = ? 27
8 TIMES  5 = ? 999
```

WORLD TIMES

WRITTEN BY RICHARD DAVIS OF MILDURA,AUSTRALIA

This program starts off by asking you to input your local time then in Line 16 subtracts 10 Hours (for eastern time).
If you live in South Australia then line 16 would read....

```
16 Z=H-9:O=M-30
```

This changes the time to greenwich time so the only line that will ever have to be changed is Line 16..(note.Line 16 has been changed for N.Z times,..Peter)
Line 20 then checks for illegal times (< zero or > 23,59) then program draws the map of the world.Extra cities can be added as follows....

```
30 E$="HAWAII":Z=GH-10:O=GM-30:C1= 3 :C2=2:X3=5:Y3=33:GOSUB 110
```

To look at this in more detail:

```
E$="HAWAII" writes HAWAII in HI-RES
Z=GH-10:O=GH-30 subtracts 10 Hours 30 Minutes from G.M.T.
C1=3:C2=2..Now C1 is the contrast color to flash the pixel.
C2 resets pixel to orignal color...
X3=5:Y3=33..X,Y Co Ordinates to flash the pixel.
```

WORLD TIMES CONTIN..

```
1 CLEAR 1000:DIME$(36):GOSUB 900:CLS
4 PRINT"  WORLD TIMES     BY PROGRAMS":PRINT:PRINT
6 PRINT"ENTER YOUR LOCAL TIME"
8 PRINT"IN 24 HOUR MODE EG..22,36":PRINT
11 INPUT H,M
12 Y1=59:MODE(1)
15 GOSUB500
16 Z=H-12:O=M
20 GOSUB 320:GH=Z:GM=O
21 Z=GH:O=GM
30 E$="HAWAII":Z=GH-10:O=GM-30:C1=4:C2=2:X3=5:Y3=33:GOSUB 110
40 E$="CAPETOWN":Z=GH+2:O=GM:C1=4:C2=2:X3=68:Y3=49:GOSUB 110
50 E$="BOMBAY":Z=GH+5:O=GM+30:C1=2:C2=4:X3=87:Y3=34:GOSUB110
55 E$="TOKYO":Z=GH+9:O=GM:C1=2:C2=4:X3=110:Y3=29:GOSUB110
60 E$="PERTH":Z=GH+8:C1=2:C2=3:X3=101:Y3=48:GOSUB110
65 E$="HONG KONG":Z=GH+8:O=GM:C1=2:C2=4:X3=100:Y3=33:GOSUB 110
70 E$="NEW ZEALAND":Z=GH-12:O=GM:C1=2:C2=3:X3=122:Y3=52:GOSUB110
75 E$="LONDON":C1=2:C2=4:X3=61:Y3=22:GOSUB110
80 E$="MOSCOW":Z=GH+2:O=GM+30:C1=2:C2=4:X3=74:Y3=21:GOSUB110
82 E$="LOS ANGELES":Z=GH-8:C1=4:C2=2:X3=18:Y3=29:GOSUB110
85 E$="NEW YORK":Z=GH-5:O=GM:C1=4:C2=2:X3=34:Y3=28:GOSUB110
90 E$="RIO DE JANEIRO":Z=GH-3:C1=2:C2=3:X3=46:Y3=45:GOSUB110
95 E$="MILDURA WERE I LIVE":Z=GH-14:O=GM:C1=2:C2=3
96 X3=112:Y3=49:GOSUB110
100 GOTO 21
110 X1=0:GOSUB320:C$=STR$(Z)
120 GOSUB130
121 C$=STR$(O):X1=X1+4
122 GOSUB130
125 GOTO 200
130 D=LEN(C$)
131 IFD=2THEN A$(0)=E$(0)ELSEGOTO135
132 N=0:GOSUB 1010:N=1:Y1=59:GOSUB1100:X1=X1+4
135 FORU=2TOD
140 E=VAL(MID$(C$,U,1))
150 N=0:A$(0)=E$(E):GOSUB1010:N=1:GOSUB1100
160 X1=X1+4:NEXT:RETURN
200 X1=32:REM LETTERS
212 L=LEN(E$)
220 FORU=1TOL
221 L$=MID$(E$,U,1)
222 Q=ASC(L$):Q=Q-55
223 IF Q<0THEN X1=X1+4:GOTO240
230 N=0:A$(0)=E$(Q):GOSUB1010:N=1:Y1=59:GOSUB1100:X1=X1+4
235 IFQ=22 OR Q=32 THENX1=X1+2
240 NEXT
250 FORI=1TO10
252 COLORC1,0:SET(X3,Y3)
254 FORN=1TO100:NEXT:SOUND1,2
256 RESET(X3,Y3)
258 FORN=1TO100:NEXT
260 NEXT:COLORC2:SET(X3,Y3)
270 COLOR1:Y=59:FORI=1TO5
272 FORN=21TO110
274 SET(N,Y):NEXT
276 Y=Y+1:NEXT
```

```
278 RETURN
320 IF O>59 THEN O=O-60:Z=Z+1
322 IF O<0 THEN O=O+60:Z=Z-1
324 IF Z<0 THEN Z=Z+24
325 IF Z>23 THEN Z=Z-24
326 RETURN
500 FORI=28672 TO 30505:READ A:POKEI,A
505 NEXT
510 RETURN
600 DATA 0,0,0,0,0,0,0,0,0,3,255,255,255,255,0,0,0,0,0,0,0,0,0,0
601 DATA 0,0,0,0,0,0,0,0
602 DATA 0,0,0,0,0,0,0,0,0,3,255,255,255,255,0,0,0,0,0,0,0,0,0,0
603 DATA 0,0,0,0,0,0,0,0,0
604 DATA 0,0,0,0,0,0,0,0,0,15,255,255,255,255,0,0,0,0,0,0,0,0,0,0
605 DATA 0,0,0,0,0,0,0,0,0
606 DATA 0,0,0,0,0,0,0,0,0,255,255,255,255,252,0,0,0,0,0,0,0,0,0,0
607 DATA 0,0,0,0,0,0,0,0,0
608 DATA 0,0,0,0,0,0,0,0,0,63,255,255,255,252,0,0,0,0,0,0,0,0,0,0
609 DATA 0,0,0,0,0,0,0,0,0
610 DATA 0,0,0,0,0,0,0,0,0,63,255,255,255,252,0,0,0,0,0,0,0,0,0,0
611 DATA 0,60,0,0,0,0,0,0,0
612 DATA 0,0,0,0,0,0,0,0,0,15,255,255,255,252,0,0,0,0,0,0,0,240
613 DATA 0,0,63,240,0,0,0,0,0,0
614 DATA 0,0,0,0,0,0,0,0,0,0,63,255,255,252,0,0,0,0,0,0,0,63,0,0
615 DATA 63,255,240,0,0,0,0,0,0
616 DATA 0,0,0,0,12,0,0,0,0,0,63,255,255,252,0,0,0,0,0,0,0,60,0,0
617 DATA 255,255,192,0,0,0,0,0,0
618 DATA 0,0,0,0,15,0,0,15,0,0,15,255,255,240,0,0,0,0,0,0,0,48,0,3
619 DATA 255,255,252,0,0,0,0,0,0
620 DATA 0,0,0,0,60,243,192,15,252,0,15,255,255,240,0,0,0,0,0,0,0
621 DATA 240,15,15,255,255,255,240,3,240,0,0,0
622 DATA 0,5,0,0,0,255,195,207,255,0,15,255,255,240,0,0,0,0,0,0,0
623 DATA 48,63,255,255,255,255,255,255,252,0,0,0
624 DATA 0,85,84,3,240,63,240,192,15,192,3,255,255,240,0,0,0,0,240
625 DATA 192,0,0,15,255,255,255,255,255,255,255,240,0,0
626 DATA 1,85,85,255,255,192,0,243,3,240,3,255,255,0,0,0,0,255
627 DATA 252,48,15,207,255,255,255,255,255,255,255,255,255,63
628 DATA 0,21,85,255,255,195,255,255,3,252,3,255,240,0,0,0,3,255
629 DATA 255,51,255,255,255,255,255,255,255,255,255,255,255,252
630 DATA 1,85,85,255,255,255,255,252,15,204,3,255,0,51,192,0,15
631 DATA 243,240,63,255,255,255,255,255,255,255,255,255,255
632 DATA 255,255
633 DATA 0,21,85,255,255,255,255,240,0,240,3,255,0,15,0,0,63,195
634 DATA 255,255,255,255,255,255,255,255,255,255,255,255,255,12
635 DATA 1,85,85,255,255,255,255,240,12,48,0,252,0,0,0,0,255,15
636 DATA 255,255,255,255,255,255,255,255,255,255,255,255,255,0
637 DATA 1,85,65,255,255,255,255,0,15,192,0,60,0,0,0,3,255,63
638 DATA 255,255,255,255,255,255,255,255,255,255,255,207,240,0
639 DATA 0,4,0,63,255,255,255,0,15,204,0,0,0,0,0,3,207,15,255
640 DATA 255,255,255,255,255,255,255,255,252,63,12,0,0
641 DATA 0,16,0,3,255,255,255,192,15,255,0,0,0,0,0,192,15,63,255
642 DATA 255,255,255,255,255,255,255,255,240,0,60,0,0
643 DATA 0,64,0,0,255,255,255,255,15,255,192,0,0,0,12,240,192,63
644 DATA 255,255,255,255,255,255,255,255,255,192,0,240,0,0
645 DATA 0,0,0,0,255,255,255,255,63,255,192,0,0,0,12,48,255,255
646 DATA 255,255,255,255,255,255,255,255,255,252,192,240,0,0
647 DATA 0,0,0,0,63,255,255,255,255,255,0,0,0,0,3,195,255,255
648 DATA 255,255,255,255,255,255,255,255,255,252,192,0,0,0
649 DATA 0,0,0,0,5,85,85,85,255,240,0,0,0,0,0,63,255,255,255
650 DATA 255,255,255,255,255,255,255,255,243,0,0,0,0
651 DATA 0,0,0,0,5,85,85,85,255,255,0,0,0,0,0,255,255,255,63
```

```
652 DATA 255,255,255,255,255,255,255,255,240,0,0,0,0
653 DATA 0,0,0,0,5,85,85,85,85,64,0,0,0,0,15,252,60,255,0,255
654 DATA 255,255,255,255,255,255,255,195,192,0,0,0
656 DATA 0,0,0,0,5,85,85,85,84,0,0,0,0,0,15,240,3,60,255,255
657 DATA 255,255,255,255,255,255,252,0,0,0,0,0
658 DATA 0,0,0,0,5,85,85,85,84,0,0,0,0,0,3,192,12,51,255,255
659 DATA 255,255,255,255,255,252,12,12,0,0,0,0
660 DATA 0,0,0,0,1,85,85,85,80,0,0,0,0,0,0,21,64,0,15,255,255
661 DATA 255,255,255,255,255,12,252,0,0,0,0
662 DATA 0,0,0,0,0,85,85,85,64,0,0,0,0,0,5,85,84,20,15
663 DATA 255,255,255,255,255,255,255,3,0,0,0,0,0
664 DATA 0,0,0,0,0,21,85,0,64,0,0,0,0,0,21,85,85,85,127,252
665 DATA 255,255,255,255,255,255,0,0,0,0,0,0
666 DATA 0,0,0,0,0,15,252,0,0,0,0,0,0,0,85,85,85,85,79,255,207
667 DATA 255,255,255,255,255,0,0,0,0,0,0
668 DATA 0,16,0,0,0,0,252,12,0,0,0,0,0,0,85,85,85,85,83,255,240
669 DATA 15,255,15,255,192,0,0,0,0,0,0
670 DATA 0,4,0,0,0,0,63,252,0,0,0,0,0,1,85,85,85,85,84,255,240
671 DATA 3,252,3,252,0,192,0,0,0,0,0
672 DATA 0,0,0,0,0,0,0,63,0,0,0,0,0,1,85,85,85,85,85,60,0,3,240
673 DATA 0,255,0,192,0,0,0,0,0
674 DATA 0,0,0,0,0,0,0,3,2,160,0,0,0,0,85,85,85,85,85,68,0,0
675 DATA 240,0,207,0,0,0,0,0,0,0
676 DATA 0,0,0,0,0,0,0,3,250,170,128,0,0,0,21,85,85,85,85,84,0
677 DATA 0,240,0,192,0,48,0,0,0,0,0
678 DATA 0,0,0,0,0,0,0,0,10,170,170,0,0,0,0,0,85,85,85,80,0,0,0
679 DATA 1,48,20,0,0,0,0,0,0,0
680 DATA 0,0,0,0,0,0,0,0,42,170,170,128,0,0,0,0,21,85,85,64
681 DATA 0,0,0,0,16,1,64,87,0,0,0,0
682 DATA 0,0,0,0,0,0,0,0,42,170,170,170,128,0,0,0,21,85,84
683 DATA 0,0,0,0,0,16,1,64,87,0,0,0,0
684 DATA 0,0,0,0,0,0,0,0,42,170,170,170,128,0,0,0,5,85,85
685 DATA 0,0,0,0,0,1,80,0,3,240,0,0,0
686 DATA 0,0,0,0,0,0,0,0,10,170,170,170,128,0,0,0,5,85,85
687 DATA 0,0,0,0,0,0,0,0,0,12,0,0,0
688 DATA 0,0,0,0,0,0,0,0,2,170,170,170,0,0,0,0,5,85,85,12,0,0
689 DATA 0,0,0,0,2,130,0,0,0,0
690 DATA 0,0,0,0,0,0,0,0,0,170,170,168,0,0,0,0,5,85,84,60,0,0,0
691 DATA 0,0,0,42,162,128,0,0,0
692 DATA 0,0,0,0,0,0,0,0,0,42,170,168,0,0,0,0,5,85,80,240
693 DATA 0,0,0,0,0,2,170,170,160,0,0,0
694 DATA 0,0,0,0,0,0,0,0,0,42,170,128,0,0,0,0,5,85,80,240
695 DATA 0,0,0,0,0,10,170,170,168,0,0,0
696 DATA 0,0,0,0,0,0,0,0,0,42,170,0,0,0,0,0,1,85,64,0,0,0,0,0,0
697 DATA 42,170,170,168,0,0,0
698 DATA 0,0,0,0,0,0,0,0,0,170,168,0,0,0,0,0,1,85,0,0,0,0,0,0,0
699 DATA 42,170,170,168,0,0,0
700 DATA 0,0,0,0,0,0,0,0,0,170,168,0,0,0,0,0,0,84,0,0,0,0,0,0,0
701 DATA 10,0,42,168,0,12,0
702 DATA 0,0,0,0,0,0,0,0,0,170,128,0,0,0,0,0,0,0,0,0,0,0,0,0,0
703 DATA 0,0,2,160,0,3,0
704 DATA 0,0,0,0,0,0,0,0,2,170,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
705 DATA 0,0,0,0,3,0
706 DATA 0,0,0,0,0,0,0,0,2,160,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
707 DATA 0,0,240,0,60,0
708 DATA 0,0,0,0,0,0,0,0,2,160,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

```
709 DATA 0,0,0,0,240,0
710 DATA 0,0,0,0,0,0,0,0,2,160,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
711 DATA 0,0,0,0,0,0
712 DATA 0,0,0,0,0,0,0,0,2,128,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
713 DATA 0,0,0,0,0,0
714 DATA 0,0,0,0,0,0,0,0,0,128,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
715 DATA 0,0,0,0,0,0
716 DATA 0,0,0,0,0,0,0,0,0,32
900 E$(0)="24444441111444444"
901 E$(1)="24111114444411111"
902 E$(2)="24414444141444414"
903 E$(3)="24414144141444444"
904 E$(4)="24444411114111444"
905 E$(5)="24444144141441444"
906 E$(6)="24444444141441444"
907 E$(7)="24411114111144444"
908 E$(8)="24444444141444444"
909 E$(9)="24444144141444444"
910 E$(10)="24144444114114444
911 E$(11)="24444444141414141
912 E$(12)="24444444111441114
913 E$(13)="24444444111414441
914 E$(14)="24444444141441414
915 E$(15)="24444444141141411
916 E$(16)="24444444111441444
917 E$(17)="24444441141144444
918 E$(18)="24411144444441114
919 E$(19)="24111441111444444
920 E$(20)="24444441141144144
921 E$(21)="24444441111411114
922 E$(22)="44444444111144444441111444444
923 E$(23)="24444444111144444
924 E$(24)="24444444111444444
925 E$(25)="24444444141144411
926 E$(26)="24444414114444441
927 E$(27)="24444444141114144
928 E$(28)="24141144141441141
929 E$(29)="24411114444441111
930 E$(30)="24444441111444444
931 E$(31)="24444411111444441
932 E$(32)="44444441111444444441111444444
933 E$(33)="24441441141144144
934 E$(34)="24444111144444411
935 E$(35)="24411444141444114
960 RETURN
1010 X=VAL(LEFT$(A$(0),1)):Y=VAL(MID$(A$(0),2,1))
1020 FOR I=0 TON:A$(I)=RIGHT$(A$(I),LEN(A$(I))-2)
1030 FOR J=0 TOX:FOR K=0 TO Y:T%(I,J,K)=VAL(LEFT$(A$(I),1))
1040 IF LEN(A$(I))=1 THEN 1060
1050 B$=RIGHT$(A$(I),LEN(A$(I))-1):A$(I)=B$
1060 NEXT:NEXT:NEXT:RETURN
1100 FOR I=0 TO Y:FOR J=0 TO X:COLOR T%(N-1,J,I)
1110 SET(X1+J,Y1+I):NEXT:NEXT:RETURN
```

SOFTWARE REVIEW

MONOPOLY
SOFTWARE MARKETED BY SCOTT LE BRUN
AND REVIEWED BY PETER J HILL (AUCKLAND N.Z)

If you have ever played the board game Monopoly,you will know that it is a excellent game which has been around for many years.And is played by many people in many countries.

Now you can play Monopoly on the VZ computer with all the frill's and deals of the board game.
For those that have never played Monopoly,its all about buying and selling real estate including railway stations,waterworks and even whole streets.

Once you start to acquire property it will really go to your head and the feel of the mighty dollar will get the better of you.
You will soon get the art of wheeling and dealing.

This program is well designed and really well written and I am sure that this VZ version will give you as much pleasure and mental stimulation as the board bound monopoly.
You are able to view your title deeds and you can even look at a Hi-Res picture of the monopoly board,which shows where each player is .Also available is a graphic picture of your present location on the board.You also have all the features of the board game such as community chest and chance.

Upto nine players can play this game,I would like to get nine people together and just see how well this great program handles all the input details.I however enjoyed playing the VZ and did Infact win one game

CONCLUSION

Another superb piece of VZ software and as stated a very excellent,and well written VZ game to play.

COST A$15-00

Available from :-
Scott Le Brun
59 Brentwood Drive
Wantirna VIC 3152
Australia

SHIFT LOCK SWITCHES BY JOE LEON

Instead of using a mechanical push on/push off switch an electronic switch can be used. Two circuits are shown using FLIP FLOPS. Which version you use will be up to you as you may allready have one or both of the IC'S.

CIRCUIT 1 :-

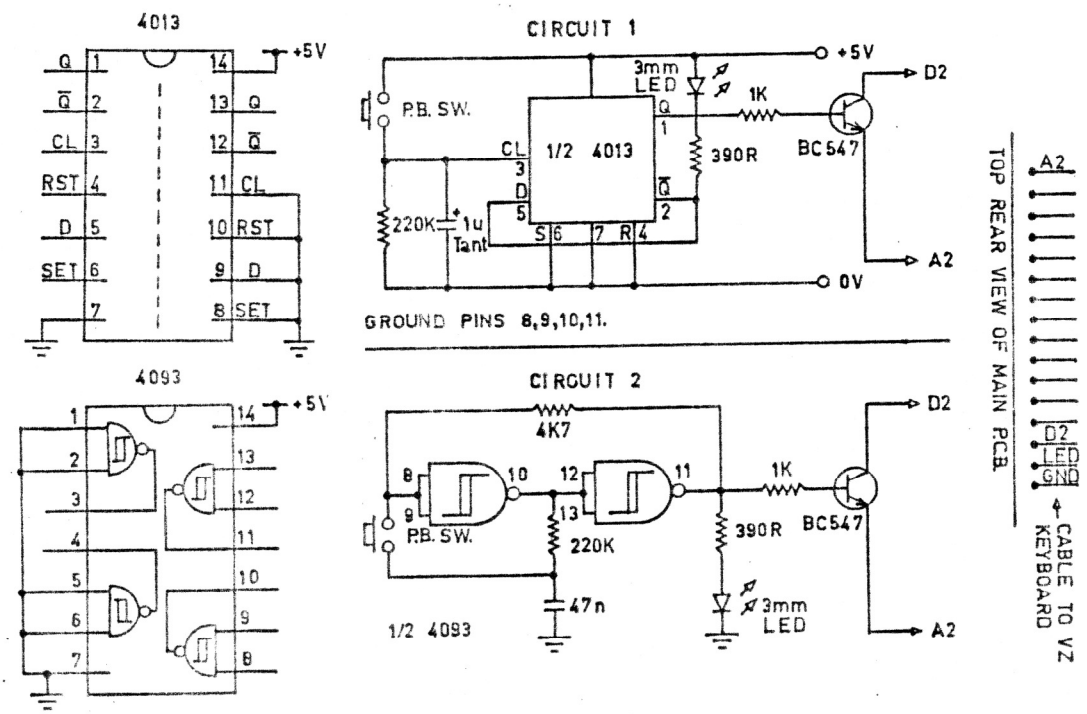
This circuit uses one half of a dual flip flop. This is the one I used. The P.B. SW. (Push Button Switch) shown can be a small momentary switch mounted anywhere convenient. In my case I used the RIGHT SHIFT KEY on the VZ 300 Keyboard. If you decide to use this option then the track on either side of the Right Shift Key on the Keyboard P.C.B. must be cut. Solder a thin wire to each side of the KEYPAD on the track leading to the isolated Shift lock switch. The other ends of the two wires go as shown on the circuit diagram. While you have the Keyboard apart you may decide to mount the 3mm LED as I did. It went in the Right Shift Key Cap itself, right above the "I" in the word SHIFT. It looks neat. The choice is yours where it goes. Connecting the Transistor to the keyboard matrix is straightforward. When built the LED will let you know when the switch is on. If the LED is on, but Shift Lock is not then try reversing the leads from the Transistor.

CIRCUIT 2 :-

This circuit does the same job as above and uses a FLIP FLOP made from NAND gates instead of a dedicated FLIP FLOP.

CAUTION :-

Both IC'S are CMOS type and special handling precautions must be observed. Do not touch the pins on the IC'S as static can destroy them. Use a socket for the IC just to be safe. Also all unused INPUTS must be grounded or taken to +5V. Pinouts for both IC'S are shown to help intending constructors. As usual take care as it's easy to make a mistake.



HIGH SCORES

Come on and send in your high scores! They aren't all that hard to beat, look even I've got some of them! Surely someone can do better?

GAME		SCORE		NAME
ASTEROIDS	-	35020	-	MATTHEW TAYLOR
CIRCUS	-	1350	-	GEOFFREY KEEN
CRASH	-	364	-	STEVEN DAVIS
DAWN PATROL	-	88200	-	STEVEN DAVIS
DEFENCE PENETRATOR	-	2820	-	SCOTT LE BRUN
DIG OUT	-	83700	-	GEOFFREY KEEN
GALAXON	-	339670	-	PETER PSARRAS
GHOST HUNTER	-	18780	-	CHRISTIAN WARNER
HAMBURGER SAM	-	99900	-	THEO HRISTAKOS
HOPPY	-	21690	-	SCOTT LE BRUN
KAMIKAZE	-	109760	-	STEVEN DAVIS
KILLER TOMATOES	-	19900	-	STEVEN DAVIS
LADDER CHALLENGE	-	25400	-	MATTHEW TAYLOR
LUNAR LANDER	-	2400	-	SCOTT LE BRUN
MISSILE COMMAND	-	20320	-	SCOTT LE BRUN
PANIK	-	16720	-	BARRY KEEN
PENGUIN	-	2670	-	SCOTT LE BRUN
PLANET PATROL	-	1701	-	SCOTT LE BRUN
ROAD WARRIOR	-	28370	-	MATTHEW TAYLOR
SPACE RAM	-	964	-	SCOTT LE BRUN
STAR BLASTER	-	835	-	STEVEN DAVIS
VZ INVADERS	-	26870	-	GEORGE SEGGIE

If you reckon you've obliterated one of the above high scores, then just send it in with your name and address, along with two witnesses signatures and you'll have yer name up in lights, right across the big country of ours and also across New Zealand!

ADVENTURE

ADVENTURER'S!! You're not neglected! If you've got some hints on how to get through an adventure, or know how to get through one completely, then send your hints in!

P.O.BOX 1972 C.P.O AUCKLAND N.Z

(C) AUCKLAND VZ 300/200 COMPUTER USER'S CLUB 1986

ALL MATERIAL IN THIS NEWSLETTER IS SUBJECT TO COPYRIGHT..
CONTRIBUTED PROGRAMS & ARTICLES CAN BE REPRODUCED ON THE UNDERSTANDING THAT THEY ARE FOR THE PRIVATE USE OF SUBSCRIBERS ONLY.
COPYRIGHT IS ALWAY'S RETAINED BY THE AUTHOR.
THIS CLUB WILL NOT BE HELD RESPONSIBLE FOR THE BREECHING OF COPYRIGHT CAUSED BY THE PUBLICATION OF ANY MATERIAL SENT IN UNDER A FALSE NAME.

YOU ARE WELCOME TO PHONE ME BETWEEN 6.30 PM AND 9 PM ----PHONE 673-967